Predicting Munich Airbnb Listing Prices

Abdessamed QCHOHI
EURECOM
Abdessamed.Qchohi@eurecom.fr

Alessio GIUFFRIDA
EURECOM
Alessio.Giuffrida@eurecom.fr

Francesco GIANNUZZO
EURECOM
Francesco.Giannuzzo@eurecom.fr

***Abstract*—This project focuses on predicting the price of an Airbnb listing in Munich based on its characteristics using machine learning and deep learning approaches. Hosts often struggle to set the right price for their listings: pricing too high can lead to low occupancy, while pricing too low may result in missed profit opportunities. Using the Inside Airbnb dataset, this project evaluates traditional regression models and neural networks to identify optimal predictive strategies. The obtained results could help Airbnb hosts understand how to price their properties fairly and attract more bookings.**

I. Introduction

Airbnb hosts face a challenge in determining optimal listing prices that maximize occupancy without undercutting profits. This task involves analyzing numerous factors, including location, amenities, and type of accommodation. Previous studies have shown that machine learning can provide useful insights into pricing dynamics. The dataset used is taken from the *Inside AirBnb* website [1], which contains information on Airbnb listings worldwide, such as location, room type, price, and amenities. The goal of this project is to leverage machine learning techniques to predict listing prices accurately. By employing both traditional regression models and more complex neural networks, we aim to identify patterns that influence pricing and offer actionable recommendations for hosts. The code of the implementation of the project can be found at the project repository [2].

II. Related work

House price prediction has evolved from traditional linear models (Rosen, 1974 [3]) to advanced machine learning techniques like Random Forests and Neural Networks, which capture non-linear patterns (Kok et al., 2017 [4], Zhan et al., 2020 [5]). Airbnb data provides insights into housing markets, influencing rental prices and capturing market trends (Horn & Merante, 2017 [6]; Barron et al., 2020 [7]). Combining Airbnb with traditional datasets enhances analysis (Koster et al., 2021 [8]). Machine learning models integrating spatial data and external factors further improve predictions (Zheng et al., 2019 [9]; Yang et al., 2021 [10]). This study leverages Airbnb data and modern machine learning methods to predict Munich house prices. A similar study on rental price prediction in the city of Munich using different machine learning methods was conducted by Chen et al. [11]. Despite the fact that we will use a different dataset, we can still use this paper as reference for our work.

III. Dataset and Features

A. Dataset Analysis

The dataset considered is in the *listings.csv.gz* file, which contains detailed listings data. While reading the file, many lines are either empty or missing some features, so they are discarded. This dataset contains 8021 entries and 75 features. We decided not to take into account the information contained in *calendar.csv.gz* and *reviews.csv.gz* because the first one has data depending on the temporal component instead of the characteristics of the accommodation, while the second one contains customer reviews, which are not directly correlated to the price.

B. Overview

Firstly we analysed the dataset and we concluded that there are a lot of useless features for the purpose of predicting listings prices: some of them include information about the host which are not relevant, others are redundant, others contain mostly missing values. Therefore, we ended up with 35 features. Then we analysed the feature *property_type*: we noticed that there are a lot of property types that appear a few times and therefore do not provide much information, so we decided to discard the ones that appear less than 35 times in the dataset.

C. Handling Missing Values

Another step is the handling of missing values. Fig. 1 illustrates the percentage of missing values in the dataset in an increasing order. The features *price* (which is the label we want to predict), *beds* and all the features regarding *review_scores* of various types are filled by grouping by *neighbourhood_cleansed* and *property_type* (in order to estimate values according to listings in the same neighbourhood and of similar type) and taking the mode for *price* and *beds*, and the mean for the different *review_scores* features. The latter (which assume float values from 1 to 5) are then combined by taking the mean, obtaining a single *review_score_mean* feature. The feature *bedrooms* is filled using as default value 1, since we assumed that all accommodations are equipped with at least one bedroom.

D. Outliers

Subsequently, we detected outliers in the *price* feature by grouping again according to *neighbourhood_cleansed* and *property_type* and computing the 99th percentile for each

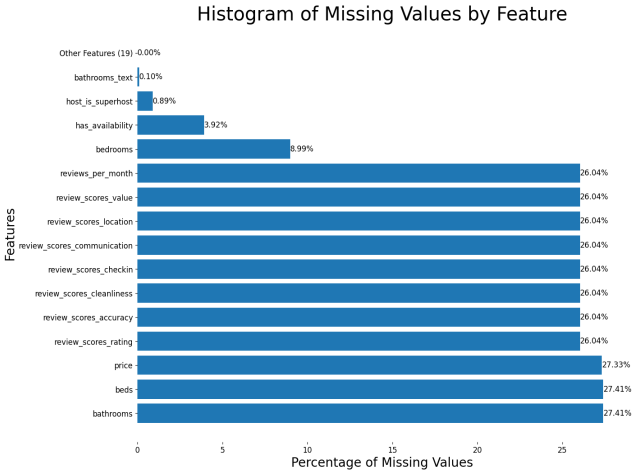

Fig. 1. Distribution of missing values

group: we discarded all rows in which the price is higher than the 99th percentile of the group.

E. Boolean and Textual Features

There are some features that express boolean information and are encoded as *t* or *f*. These are mapped to proper boolean values *True* or *False*. If there are missing values in these features they are filled using *False* as a default value (this is the case for *host_is_superhost*). An exception is the feature *has_availability*, which is treated in the following way: if the value of the *availability_30* feature, which represents the number of days the accommodation is available in the next 30 days, is greater or equal than 25, then we assigned *True*, otherwise *False*. The features *number_of_reviews_ltm* and *number_of_reviews_l30d* contain many 0 values, so they are transformed to boolean values by checking if the value is greater than 0. There are also features that represent textual information, and according to their meaning they are treated in different ways:

- *bathrooms_text* contain a textual description of the number and the type of bathroom: since the feature *bathrooms*, which indicates the number of bathrooms, has many more missing values than *bathrooms_text*, we decided to extract from *bathrooms_text* the information about the number and the type separately, creating two new features *type_bathroom* and *num_bathrooms*, and then we compared *bathrooms* and *num_bathrooms* to fill missing values in *num_bathrooms* (we discard rows where both the values are missing). Finally, we dropped the original features *bathrooms* and *bathrooms_text*.
- the *amenities* feature contains a list of all the amenities of an accommodation: since there a lot of different possible values, we extracted the amenities that according to our evaluation contribute the most to the price (like TV, Wifi, Bluetooth...) and assigned *True*, if they are present, or *False*, otherwise.

We also tried some basic sentiment analysis using the *description* feature, but later on during the training of the models we did not obtain any significant improvements, so we decided to remove this feature. We then performed one-hot encoding of the categorical features *neighbourhood_cleansed*, *property_type* and *type_bathroom*, and we normalized numerical features by using *StandardScaler*. Finally, we obtained a dataset with 7391 entries and 80 features.

IV. Methods

The task involves solving a regression problem. To address it, we adopted two approaches: traditional regression models and more complex neural networks. The models we chose are:

- Lasso and Ridge: they implement linear regression using L1 and L2 regularization, respectively. These two models serve as a baseline that we try to improve upon.
- Random Forest: it is an ensemble method that fits a number of decision tree regressors on various subsamples of the dataset and uses averaging to improve the predictive accuracy and control overfitting.
- Gradient Boosting: this method builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage a regression tree is fit on the negative gradient of the given loss function.
- Stacking Regressor: it is a model that combines the predictions of other regression models to produce a more robust and accurate final prediction. We decided to build it with the best Random Forest and Gradient Boosting models found.
- Multi-Layer Perceptron (MLP): a feedforward neural network consisting of fully connected neurons with nonlinear activation functions, organized in layers.
- Deep neural network: a neural network that consists of multiple layers of interconnected nodes, also known as neurons. These networks are considered ”deep” because they have more than one hidden layer between the input and output layers, enabling them to model complex patterns and relationships in data.
- Deep neural network with hypernetwork [12]: a hypernetwork is a type of neural network designed to generate the weights of another neural network. In essence, it is a higher-level network that outputs the parameters (weights and biases) for a target neural network, enabling dynamic and flexible weight generation.

V. Experiments, Results and Discussion

Firstly, we split the dataset into a training and test set using a 80/20 split. Then we performed K-fold cross validation for all the models considered (except for Lasso and Ridge regression and the deep neural network). After that we tested the models and evaluated their R^2 score and root mean squared error (RMSE). These metrics are useful to evaluate the quality of a regression model: the R^2 score is a measure of the goodness of fit of a model, while the RMSE is the quadratic mean of the differences between the observed values and predicted ones. For Lasso and Ridge regression we created two simple models using the default values of the hyperparameters. These

two models serve as our baseline. About Random Forest and Gradient Boosting we performed a cross validation with 3 folds on the hyperparameters in Tables I and II. The results obtained are remarkably good compared to the baseline.

TABLE I
HYPERPARAMETERS RANDOM FOREST

Model	Parameter	Values
Random Forest	n_estimators	{150, 300, 450}
	max_depth	{None, 10, 30}
	min_samples_split	{2, 5, 10}
	min_samples_leaf	{1,2,4}

TABLE II
HYPERPARAMETERS GRADIENT BOOSTING

Model	Parameter	Values
Gradient Boosting	n_estimators	{100, 300, 450}
	max_depth	{4, 6, 8}
	learning_rate	{0.01, 0.05, 0.1}
	subsample	{0.8, 0.9, 1.0}
	min_samples_leaf	{1, 3, 5}

We also analysed the feature importances returned by the random forest and we noticed that many features have low importance. We therefore decided to remove the features with an importance value lower than 0.004 (Fig. 2 shows what are these features) and tried to train and test again the model. Despite this adjustment, the performance is almost the same.

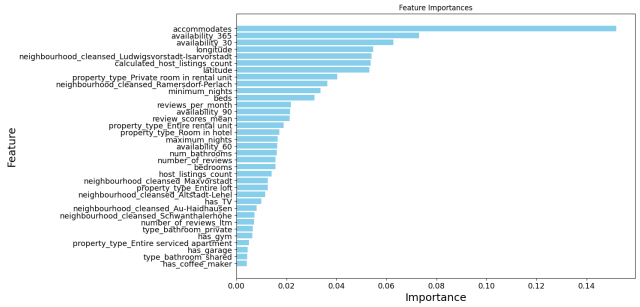

Fig. 2. Feature importances of most important features

Finally, we combined the two models in a Stacking Regressor, still performing a K-fold cross-validation with 3 folds. We used the best combination of hyperparameters found from the previous tuning of Random Forest and Gradient Boosting, performing a final tuning for the Stacking Regressor (Table III).

TABLE III
HYPERPARAMETERS STACKING REGRESSOR

Model	Parameter	Values
Stacking Regressor	n_estimators	{100, 300}
	max_depth	{4, 6, 8}
	learning_rate	{0.01, 0.05, 0.1}
	max_depth	{3, 5, 7}

We followed the same approach for the Multi-Layer Perceptron, performing a cross validation with 5 folds on the hyperparameters in Table IV.

TABLE IV
HYPERPARAMETERS MULTI-LAYER PERCEPTRON

Model	Parameter	Values
Multi-Layer Perceptron	hidden_layer_sizes	{(100), (100, 100), (100, 100, 100)}
	activation	{relu, tanh}
	solver	{lbfgs, adam }
	alpha	{0.0001, 0.001}
	learning_rate	{adaptive, constant}
	learning_rate_init	{0.0001, 0.001}
	max_iter	{200}

Next we used a deep neural network: after some initial trials, we found that a more complex architecture is better at representing the relationships between the features of the dataset. In particular, between the layers we used a state-of-the-art activation function called Mish, which is mathematically defined as $f(x) = x \cdot tanh(softplus(x))$ [13]. We also used a dropout layer that randomly sets to 0 some of the elements of the input tensor with a certain probability during training time, which helps prevent overfitting, and a batch normalization layer that makes training more stable through normalization of the layers' inputs by re-centering and re-scaling. As for the optimizer, we used RMSprop with $\alpha = 0.9$ and $\epsilon = 1e-16$. We also used the $weight\_decay$ hyperparameter to prevent overfitting of the network during training. Before doing the train/test split, we used a random forest regressor to compute the features importances and to remove the features with a score less than 0.002: in this way we wanted to keep only the most informative features to reduce noise and improve performance. We performed a grid search for 500 epochs on the hyperparameters in Table V.

TABLE V
HYPERPARAMETERS DEEP NEURAL NETWORK

Model	Parameter	Values
Deep Neural Network	learning_rate	{0.001, 0.01}
	weight_decay	{1e-8, 1e-6, 1e-4}
	dropout	{0.4, 0.5, 0.6}
	batch_size	{64, 128, 256}

The results of this tuning can be seen in Fig. 3 Then we chose the best 5 configurations with the lowest RMSE and highest R^2 score (they had similar performance) and we performed another tuning, this time for 1500 epochs, obtaining the best configuration of hyperparameters along with its performance. Since from Fig. 4 and Fig. 5 we can see that the training of the network with these hyperparameters led to a stable behavior, we concluded that it was not necessary to re-train with a higher number of epochs. From the figures below we can see that there is not much overfitting, as the difference between the training and the test curves is not so large. Overall, we can conclude that the results obtained are comparable with those of the other models.

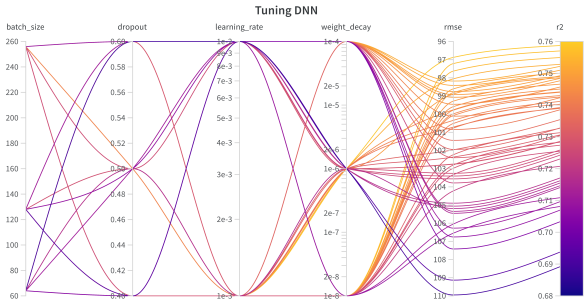

Fig. 3. Tuning of hyperparameters of deep neural network

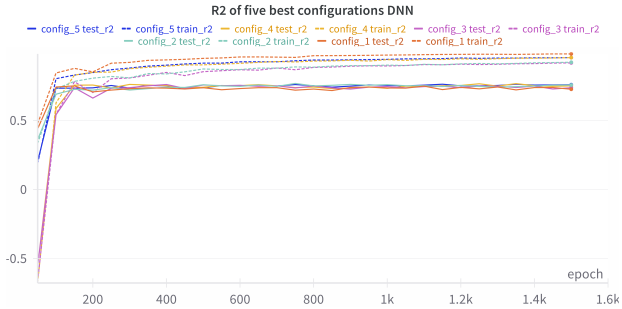

Fig. 4. R^2 score of the five best configurations for DNN

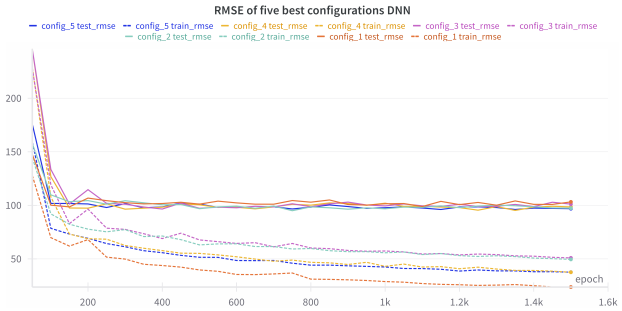

Fig. 5. RMSE of the five best configurations for DNN

Finally, we tried to improve this deep learning approach by implementing a deep neural network with hypernetworks: their goal is to dynamically generate weights for layers in the primary network based on an input latent vector. We defined an Embedding and a HyperNetwork class that serve this purpose. The main network defines two hypernetworks and some static layers which represent a portion of the neural network we defined previously. During the forward pass the first hypernetwork uses an embedded vector to generate weights for the first linear layer. After applying the weights to the input using the linear layer, we apply batch normalization, activation function and dropout. Then a similar process is repeated for the second layer with another hypernetwork on another embedded vector. Finally, the processed data is passed through the static output layers. We also tried to use more than two hypernetworks, but this led to a more unstable behavior during training. For the new network we performed a grid search for 1000 epochs on the hyperparameters in Table VI. To cope with the addition of the hypernetworks, we had to reduce the complexity of the static part of the main network by removing some layers, which led to a significant increase in performance. Exactly as we did for the simple deep neural network, we took the five best performing configurations and performed another tuning phase, this time with 3000 epochs, to try to reach convergence. The trend of all configurations derived from the first tuning phase can be found in Fig. 6, instead the results of the best five configurations are shown in Fig. 7 and Fig. 8.

TABLE VI
HYPERPARAMETERS DEEP NEURAL NETWORK WITH HYPERNETWORK

Model	Parameter	Values
Deep Neural Network with HyperNetwork	learning_rate	{0.001, 0.01}
	weight_decay	{1e-8, 1e-6, 1e-4}
	dropout	{0.4, 0.5, 0.6}
	batch_size	{64, 128, 256}

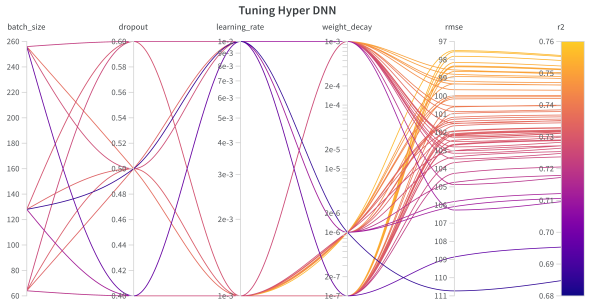

Fig. 6. Tuning of hyperparameters of deep neural network with hypernetworks

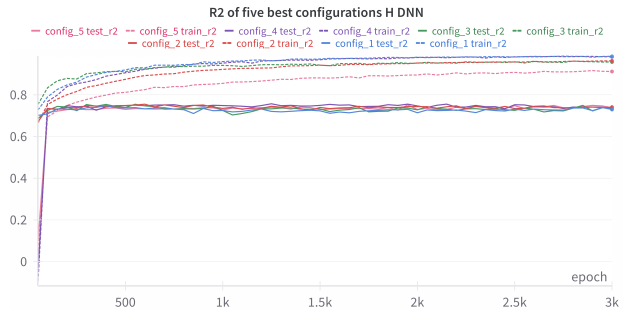

Fig. 7. R^2 score of the five best configurations for DNN with hypernetworks

Looking at the performance data, it is evident that the addition of two hypernetworks did not introduce a significant improvement in the primary network. All configurations reach a plateau for the values of the metrics during test time, leading to slight overfitting. We believe that these behaviors are due to the considerable level of complexity added to the network, since the network adapts too well to the noise present in the

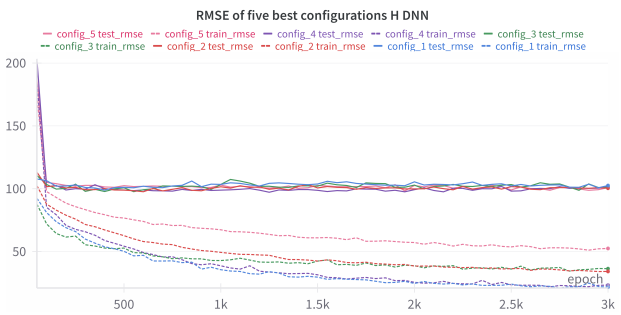

Fig. 8. RMSE of the five best configurations for DNN with hypernetworks

data, leading to a worse generalization capability. All the best hyperparameters obtained and the results are summarised in Tables VII and VIII.

TABLE VII
MODELS HYPERPARAMETERS

Model	Best Hyperparameters
Random Forest	{'max_depth': 30, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300}
Gradient Boosting	{'learning_rate': 0.05, 'max_depth': 6, 'min_samples_leaf': 3, 'n_estimators': 450, 'subsample': 0.8}
Stacking Regressor	{'learning_rate': 0.05, 'max_depth': 3, 'n_estimators': 100}
Multi-Layer Perceptron	{'hidden_layer_sizes': (100, 100), 'activation': 'relu', 'solver': 'adam', 'alpha': 0.0001 , 'learning_rate': 'constant', 'learning_rate_init': 0.001, 'max_iter': 200 }
Deep Neural Network	{'learning_rate': 0.001, 'weight_decay': 1e-6, 'dropout': 0.5, 'batch_size': 256}
Deep Neural Network with HyperNetworks	{'learning_rate': 0.01, 'weight_decay': 1e-7, 'dropout': 0.5, 'batch_size': 64}

TABLE VIII
RESULTS

Model	R^2 score	RMSE
Lasso regression	0.367	157.03
Ridge regression	0.415	150.78
Random Forest	0.748	99.08
Random Forest with only most important features	0.748	99.10
Gradient Boosting	0.769	94.77
Stacking Regressor	0.779	92.76
Multi-Layer Perceptron	0.714	105.45
Deep Neural Network	0.776	93.33
Deep Neural Network with HyperNetworks	0.763	96.07

VI. CONCLUSIONS AND FUTURE WORK

We came to the conclusion that using a deep neural network we could not obtain significant improvement with respect to the traditional regression models. Moreover, adding the hypernetwork led to slightly worse results, likely indicating that the available data is not enough for such a complex model to learn from. During the pre-processing phase we made many assumptions and choices to try to retain as many features of the original data as possible, but we had to work with a very dirty dataset with many missing values: this led to a total number of records that is not ideal for training a model for a regression problem of this complexity. As future improvements, it could be possible to implement advanced sentiment analysis, to try to exploit the information contained in the *reviews.csv.gz* dataset. It could also be useful to resort to synthetic data to have a larger dataset to work with.

VII. CONTRIBUTIONS

In this section we will describe the work done by each team member. Data exploration and preprocessing were carried out by everyone together. We also decided together on the models used to tackle the problem. Abdessamed focused on the Random Forest and on the possibility of using only the most important features on the basis of the importance assigned by the model. Francesco implemented Gradient Boosting and the Stacking Regression obtained from the combination of Random Forest and Gradient Boosting. Alessio implemented the Ridge and Lasso regression used as baseline and the Multi-Layer Perceptron. Finally, we all worked together on the Deep neural network and the Hypernetwork implementations. As for LLM usage, we used ChatGPT to help us write the report and GitHub Copilot to assist us in writing the code.

REFERENCES

[1] Inside Airbnb dataset. https://insideairbnb.com/get-the-data/.
[2] Abdessamed Qchohi, Alessio Giuffrida, and Francesco Giannuzzo, "MALIS Predicting Munich Airbnb Prices". https://github.com/franceth/-MALIS-Predicting-Munich-Airbnb-Prices
[3] S. Rosen, "Hedonic Prices and Implicit Markets: Product Differentiation in Pure Competition", 1974. https://matthewturner.org/ec2410/readings/Rosen_JPE_1974.pdf
[4] N. Kok, E.-L. Koponen, C. A. Martínez-Barbosa, "Big Data in Real Estate? From Manual Appraisal to Automated Valuation", 2017. https://sustainable-finance.nl/upload/researches/Kok-et-al_Big-Data-in-Real-Estate.pdf
[5] C. Zhan, Z. Wu, Y. Liu, Z. Xie, W. Chen, "Housing prices prediction with deep learning: an application for the real estate market in Taiwan", 2020. https://ieeexplore.ieee.org/document/9442244
[6] K. Horn, M. Merante, "Is home sharing driving up rents? Evidence from Airbnb in Boston", 2017. https://www.sciencedirect.com/science/article/pii/S1051137717300876
[7] K. Barron, E. Kung, D. Proserpio, "The Effect of Home-Sharing on House Prices and Rents: Evidence from Airbnb", 2020. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3006832
[8] H. R. A. Koster, J. van Ommeren, N. Volkhausen, "Short-term rentals and the housing market: Quasi-experimental evidence from Airbnb in Los Angeles", 2021. https://www.sciencedirect.com/science/article/pii/S0094119021000383
[9] Z. Zheng, S. Yu, M. Li, K. Zhang, M. Zhu, Y. He, Q. Peng, "Mass Appraisal of Urban Housing Based on GIS and Deep Learning", 2023. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4510568
[10] Z. Yang, X. Zhu, Y. Zhang, P. Nie, X. Liu, "A Housing Price Prediction Method Based on Stacking Ensemble Learning Optimization Method", 2023.https://ieeexplore.ieee.org/document/10195626
[11] W. Chen, S. Farag, U. Butt, H. Al-Khateeb, "Leveraging Machine Learning for Sophisticated Rental Value Predictions: A Case Study from Munich, Germany", 2024. https://www.mdpi.com/2076-3417/14/20/9528

[12] HyperNetwork implementation.
https://github.com/g1910/HyperNetworks
[13] D. Misra, "Mish: A Self Regularized Non-Monotonic Activation Function", 2019. https://arxiv.org/pdf/1908.08681